(2)

REPORT DOCUMENTATION PAGE

Form Approved
OPM No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		Final 27 Oct. 1989 to 27 Oct. 1990

4. TITLE AND SUBTITLE Ada Compiler Validation Summary Report: DDC INTERNATIONAL A/S, DACS-386/UNIX, Version 4.4, RC900 (Host & Target), 891027S1.10186

5. FUNDING NUMBERS

6. AUTHOR(S)
National Institute of Standards and Technology
Gaithersburg, MD
USA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
National Institute of Standards and Technology
National Computer Systems Laboratory
Bldg. 255, Rm. A266
Gaithersburg, MD 20899
USA

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

DDC INTERNATIONAL A/S DACS-386/UNIX, Version 4.4, Gaithersburg MD, RC900 under UNIX V, Release 1.0.4 (Host & Target), ACVC 1.10.

14. SUBJECT TERMS Ada programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, Validation Testing, Ada Validation Office, Ada Validation Facility, ANSI/MIL-STD-1815A, Ada Joint Program Office

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	

AVF Control Number: NIST89DDC580_4_1.10
DATE VSR COMPLETED BEFORE ON-SITE: 10-02-89
DATE VSR COMPLETED AFTER ON-SITE: 10-30-89
DATE VSR MODIFIED PER AVO COMMENTS: 12-14-89
DATE VSR MODIFIED PER AVO COMMENTS: 04-30-90

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 891027S1.10186
DDC INTERNATIONAL A/S
DACS-386/UNIX, Version 4.4
RC900 Host and RC900 Target

Completion of On-Site Testing:
27 October 1989

Prepared By:
Software Standards Validation Group
National Computer Systems Laboratory
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

AVF Control Number: NIST89DDC580_4_1.10
DATE COMPLETED BEFORE ON-SITE: 10-02-89
DATE COMPLETED AFTER ON-SITE: 10-30-89

Ada Compiler Validation Summary Report:

Compiler Name: DACS-386/UNIX, Version 4.4

Certificate Number: 891027S1.10186

Host: RC900 under UNIX V, Release 1.0.4

Target: RC900 under UNIX V, Release 1.0.4

Testing Completed October 27, 1989 Using ACVC 1.10

This report has been reviewed and is approved.

Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division
National Computer Systems
Laboratory (NCSL)
National Institute of
Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899

Ada Validation Facility
Mr. L. Arnold Johnson
Manager, Software Standards
Validation Group
Engineering Division
National Computer Systems
Laboratory (NCSL)
National Institute of
Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr. John Solomond
Director
Department of Defense
Washington DC 20301

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.
The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent, but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 27 October 1989 at Lyngby, Denmark.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Software Standards Validation Group
National Computer Systems Laboratory
National Institue of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.
3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.
4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC — The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.

Ada Commentary — An Ada Commentary contains all information relevant to the Commentary point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.

Ada Standard — ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

Applicant — The agency requesting validation.

AVF — The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures and Guidelines.

AVO — The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and

technical support for Ada validations to ensure consistent practices.

Compiler — A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.

Failed test — An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host — The computer on which the compiler resides.

Inapplicable test — An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.

Passed test — An ACVC test for which a compiler generates the expected result.

Target — The computer which executes the code generated by the compiler.

Test — A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.

Withdrawn — An ACVC test found to be incorrect and not used to check test conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation and execution of legal Ada programs with certain language constructs which cannot be verified at run time. There are no explicit program components in a Class A test

to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED,

FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for Chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: DACS-386/UNIX, Version 4.4

ACVC Version: 1.10

Certificate Number: 891027S1.10186

Host Computer:

Machine: RC900

Operating System: UNIX V, Release 1.0.4

Memory Size: 16 MBytes

Target Computer:

Machine: RC900

Operating System: UNIX V, Release 1.0.4

Memory Size: 16 MBytes

Communications Network:

From VAX-8530 to RC900:

VAX-8530 via Ethernet (using DNICP net software utility) via SUN-3 Workstation via streamer tape to the RC900 (386/UNIX V Workstation) using UNIX tar format with block = 20.

From RC900 to VAX-8530:

TWO METHODS were used depending upon availability of the

software/hardware: (1) kermit via terminal hardware connection to the VAX-8530; (2) the files on the RC900 are written to a UNIX tar file which is then fragmented into 360KByte groups which are then written to DOS 360 KByte diskettes; the DOS 360 KBytes diskettes are loaded onto an ICL DRS300 where the 360KByte file groups are concatenated to form the original tar file; the original tar file is then copied to a streamer tape which is then loaded onto the Sun-1 Workstation or Sun-3 Workstation; from the Sun-1 or the Sun-3 the file(s) are transferred by Ethernet (using DNICP net software utility) to the VAX-8530.

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

a. Capacities.

(1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)

(2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests).)

(3) The compiler rejects tests containing block statements nested to 65 levels. (See test D56001B.)

(4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 17 levels. (See tests D64005E..G (3 tests).)

b. Predefined types.

(1) This implementation supports the additional predefined types SHORT_INTEGER, LONG_FLOAT, and LONG_INTEGER in the package STANDARD. (See tests B86001T..Z (7 tests).)

c. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate

the following:

(1) All of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)

(2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

(3) This implementation uses no extra bits for extra precision and uses all extra bits for extra range. (See test C35903A.)

(4) NUMERIC_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

(5) NUMERIC_ERROR is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

(6) Underflow is gradual. (See tests C45524A..K (11 tests).)

d. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

(1) The method used for rounding to integer is round to even. (See tests C46012A..K (11 tests).)

(2) The method used for rounding to longest integer is round to even. (See tests C46012A..K (11 tests).)

(3) The method used for rounding to integer in static universal real expressions is round away from zero. (See test C4A014A.)

e. Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT. For this implementation:

(1) Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises NUMERIC_ERROR.

(See test C36003A.)

(2) NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components. (See test C36202A.)

(3) NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components. (See test C36202B.)

(4) A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when declaring two packed Boolean arrays with INTEGER'LAST + 3 components. (See test C52103X.)

(5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array type is declared. (See test C52104Y.)

(6) A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

(7) In assigning one-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

(8) In assigning two-dimensional array types, the expression is not evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

f. Discriminated types.

(1) In assigning record types with discriminants, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

g. Aggregates.

(1) In the evaluation of a multi-dimensional aggregate, the test results indicate that all choices are evaluated before checking against the index type. (See tests C43207A and

C43207B.)

(2) In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)

(3) CONSTRAINT_ERROR is raised before all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

h. Pragmas.

(1) The pragma INLINE is supported for functions or procedures. (See tests LA3004A..B (2 tests), EA3004C..D (2 tests), and CA3004E..F (2 tests).)

i. Generics.

(1) Generic specifications and bodies cannot be compiled in separate compilations. (See tests CA2009C, CA2009F, BC3204C, and BC3205D.)

Generic package declarations and bodies can be compiled in separate compilations so long as no instantiations of those units precede the bodies. This compiler requires that a generic unit's body be compiled prior to instantiation, and so the unit containing the instantiations is rejected.

(2) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

(3) Generic subprogram declarations and bodies can be compiled in separate compilations. (See test CA1012A.)

(4) Generic library subprogram specifications and bodies can be compiled in separate compilations. (See test CA1012A.)

(5) Generic non-library subprogram bodies cannot be compiled in separate compilations from their stubs. (See test CA2009F.)

(6) Generic package declarations and bodies cannot be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

(7) Generic library package specifications and bodies cannot be compiled in separate compilations. (See tests BC3204C and BC3205D.)

(8) Generic non-library package bodies as subunits cannot be

compiled in separate compilations. (See test CA2009C.)

(9) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

j. Input and output.

(1) The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D and EE2201E.)

(2) The package DIRECT_IO can be instantiated with unconstrained array types but only if the maximum element size supported for DIRECT_IO is 2_147_483_647 bits; otherwise, USE_ERROR is raised. (See tests AE2101H and EE2401D.)

(3) The package DIRECT_IO can be instantiated with record types with discriminants without defaults. (See test EE2401G.)

(4) USE_ERROR is raised when Mode IN_FILE is not supported for the operation of CREATE for SEQUENTIAL_IO. (See test CE2102D.)

(5) USE_ERROR is raised when Mode IN_FILE is not supported for the operation of CREATE for DIRECT_IO. (See test CE2102I.)

(6) USE_ERROR is raised when Mode IN_FILE is not supported for the operation of CREATE for text files. (See test CE3102E.)

(7) Modes IN_FILE and OUT_FILE are supported for text files. (See test CE3102I..K).

(8) RESET and DELETE operations are supported for SEQUENTIAL_IO. (See tests CE2102G and CE2102X.)

(9) RESET and DELETE operations are supported for DIRECT_IO. (See tests CE2102K and CE2102Y.)

(10) RESET and DELETE operations are supported for text files. (See tests CE3102F..G (2 tests), CE3104C, CE3110A, and CE3114A.)

(11) Overwriting to a sequential file truncates to the last element written. (See test CE2208B.)

(12) Temporary sequential files are given names and deleted when closed. (See test CE2108A.)

(13) Temporary direct files are given names and deleted when closed. (See test CE2108C.)

(14) Temporary text files are given names and deleted when closed. (See test CE3112A.)

(15) More than one internal file can be associated with each external file for sequential files when writing or reading. (See tests CE2107A..E (5 tests), CE2102L, CE2110B, and CE2111D.)

(16) More than one internal file can be associated with each external file for direct files when writing or reading. (See tests CE2107F..H (3 tests), CE2110D and CE2111H.)

(17) More than one internal file can be associated with each external file for text files when writing or reading. (See tests CE3111A, CE3111D..E (2 tests), and CE3114B.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 44 tests had been withdrawn because of test errors. The AVF determined that 433 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for 74 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	123	1132	1897	16	26	46	3240
Inapplicable	6	6	418	1	2	0	433
Withdrawn	1	2	35	0	6	0	44
TOTAL	130	1140	2350	17	34	46	3717

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	195	572	554	247	172	99	161	331	135	36	250	191	297	3040
Inapplicable	17	77	126	1	0	0	5	1	2	0	2	178	24	433
Wdrn	1	1	0	0	0	0	0	2	0	0	1	35	4	44
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717

3.4 WITHDRAWN TESTS

The following 44 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

A39005G	B97102E	C97116A	BC3009B	CD2A62D	CD2A63A
CD2A63B	CD2A63C	CD2A63D	CD2A66A	CD2A66B	CD2A66C
CD2A66D	CD2A73A	CD2A73B	CD2A73C	CD2A73D	CD2A76A
CD2A76B	CD2A76C	CD2A76D	CD2A81G	CD2A83G	CD2A84M
CD2A84N	CD2B15C	CD2D11B	CD5007B	CD5011O	CD7105A
CD7203B	CD7204B	CD7205C	CD7205D	CE2107I	CE3111C
CE3301A	CE3411B	E28005C	ED7004B	ED7005C	ED7005D
ED7006C	ED7006D				

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 433 tests were inapplicable for the reasons indicated:

a. The following 201 tests are not applicable because they have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113L..Y (14 tests) C35705L..Y (14 tests)
C35706L..Y (14 tests) C35707L..Y (14 tests)

C35708L..Y (14 tests) C35802L..Z (15 tests)
C45241L..Y (14 tests) C45321L..Y (14 tests)
C45421L..Y (14 tests) C45521L..Z (15 tests)
C45524L..Z (15 tests) C45621L..Z (15 tests)
C45641L..Y (14 tests) C46012L..Z (15 tests)

b. C24113I..K (3 tests) are not applicable because the line length of the input file must not exceed 126 characters.

c. C35508I, C35508J, C35508M, C35508N, AD1C04D, AD3015C, AD3015F, AD3015H, AD3015K, CD1C04B, CD1C04C, CD1C04E, CD2A23C, CD2A23D, CD2A24C, CD2A24D, CD2A24G, CD2A24H, CD3015A, CD3015B, CD3015D, CD3015E, CD3015G, CD3015I, CD3015J, CD3015L, CD4051A, CD4051B, CD4051C, CD4051D (30 tests) are not applicable because this implementation does not support the specified change in representation for derived types.

d. C35702A and B86001T are not applicable because this implementation supports no predefined type SHORT_FLOAT.

e. A39005E, C87B62C, CD1009L, CD1C03F, CD2D11A, CD2D13A, ED2A56A (7 tests) are not applicable because 'SMALL clause is not supported.

f. C45231D, CD7101G, and B86001X, are not applicable because this implementation does not support any predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER.

g. C45531M, C45531N, C45532M, and C45532N use fine 48 bit fixed point base types which are not supported by this compiler.

h. C45531O, C45531P, C45532O, and C45532P use coarse 48 bit fixed point base types which are not supported by this compiler.

i. C4A013B is not applicable because the evaluation of an expression involving 'MACHINE_RADIX applied to the most precise floating-point type would raise an exception; since the expression must be static, it is rejected at compile time.

j. D56001B uses 65 levels of block nesting which exceeds the capacity of the compiler.

k. B86001Z is not applicable because this implementation supports no predefined floating-point type with a name other than FLOAT, or LONG_FLOAT.

l. B86001Y is not applicable because this implementation supports no predefined fixed-point type other than DURATION.

m. C96005B is not applicable because there are no values of type DURATION'BASE that are outside the range of DURATION.

n. CA2009C is not applicable because this implementation does not

permit compilation of generic non-library package bodies in separate files from their specifications.

o. CA2009F is not applicable because this implementation does not permit compilation of generic non-library subprogram bodies in separate files from their specifications.

p. BC3204C and BC3205D are not applicable because this implementation does not permit compilation of generic library package bodies in separate files from their specifications.

q. CD1009C, CD2A41A..B, CD2A41E, and CD2A42A..J (14 tests) are not applicable because this implementation does not support the 'SIZE clause for floating-point types.

r. CD2A51C, CD2A52A..D, CD2A52G..J, CD2A53A..E, CD2A54A..D, CD2A54G..J (22 tests) are not applicable because this implementation does not support the 'SIZE clause for a fixed-point types.

s. CD2A61A..L, CD2A62A..C, CD2A64A..D, CD2A65A..D, CD2A71A..D, CD2A72A..D, CD2A74A..D, CD2A75A..D (39 tests) are not applicable because this implementation does not support the 'SIZE clause for an array type which does not imply compression of inter-component gaps.

t. CD2A84B..I and CD2A84K..L (10 tests) are not applicable because this implementation does not support the SIZE clause other than the default value for an access type.

u. CD4041A is not applicable because this implementation does not support the alignment clauses for alignments other than SYSTEM.STORAGE_UNIT for record representation clauses.

v. CD5003B..I, CD5011A, CD5011C, CD5011E, CD5011G, CD5011I, CD5011K, CD5011M, CD5011Q, CD5012A..B, CD5012E..F, CD5012I, CD5012M, CD5013A, CD5013C, CD5013E, CD5013G, CD5013I, CD5013K, CD5013M, CD5013O, CD5014T, CD5014V..Z (36 tests) are not applicable because this implementation does not support non-static address clauses for a variable.

w. CD5011B, CD5011D, CD5011F, CD5011H, CD5011L, CD5011N, CD5011R, CD5011S, CD5012C..D, CD5012G..H, CD5012L, CD5013B, CD5013D, CD5013F, CD5013H, CD5013L, CD5013N, CD5013R, CD5014U (21 tests) are not applicable because this implementation does not support non-static address clauses for a constant.

x. CD5012J, CD5013S, CD5014S (3 tests) are not applicable because this implementation does not support non-static address clauses.

y. CE2102E is inapplicable because this implementation supports CREATE with OUT_FILE mode for SEQUENTIAL_IO.

z. CE2102F is inapplicable because this implementation supports CREATE with INOUT_FILE mode for DIRECT_IO.

aa. CE2102J is inapplicable because this implementation supports CREATE with OUT_FILE mode for DIRECT_IO.

ab. CE2102N is inapplicable because this implementation supports OPEN with IN_FILE mode for SEQUENTIAL_IO.

ac. CE2102O is inapplicable because this implementation supports RESET with IN_FILE mode for SEQUENTIAL_IO.

ad. CE2102P is inapplicable because this implementation supports OPEN with OUT_FILE mode for SEQUENTIAL_IO.

ae. CE2102Q is inapplicable because this implementation supports RESET with OUT_FILE mode for SEQUENTIAL_IO.

af. CE2102R is inapplicable because this implementation supports OPEN with INOUT_FILE mode for DIRECT_IO.

ag. CE2102S is inapplicable because this implementation supports RESET with INOUT_FILE mode for DIRECT_IO.

ah. CE2102T is inapplicable because this implementation supports OPEN with IN_FILE mode for DIRECT_IO.

ai. CE2102U is inapplicable because this implementation supports RESET with IN_FILE mode for DIRECT_IO.

aj. CE2102V is inapplicable because this implementation supports OPEN with OUT_FILE mode for DIRECT_IO.

ak. CE2102W is inapplicable because this implementation supports RESET with OUT_FILE mode for DIRECT_IO.

al. CE2105A is inapplicable because CREATE with IN_FILE mode is not supported by this implementation for SEQUENTIAL_IO.

am. CE2105B is inapplicable because CREATE with IN_FILE mode is not supported by this implementation for DIRECT_IO.

an. CE3102F is inapplicable because text file RESET is supported by this implementation.

ao. CE3102G is inapplicable because text file deletion of an external file is supported by this implementation.

ap. CE3102I is inapplicable because text file CREATE with OUT_FILE mode is supported by this implementation.

aq. CE3102J is inapplicable because text file OPEN with IN_FILE mode is

supported by this implementation.

ar. CE3102K is inapplicable because text file OPEN with OUT_FILE mode is supported by this implementation.

as. CE3109A is inapplicable because text file CREATE with IN_FILE mode is not supported by this implementation.

at. CE3111B and CE3115A simultaneously associate input and output files with the same external file, and expect that output is immediately written to the external file and available for reading; this implementation buffers files, and each test's attempt to read such output (at lines 87 & 101, respectively) raises END_ERROR.

au. EE2401D is inapplicable because the maximum element size supported for DIRECT_IO is 2_147_483_647 bits. USE_ERROR is raised.

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that was not anticipated by the test (such as raising one exception instead of another).

Modifications were required for 74 tests.

The following 65 tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B22003A	B26001A	B26002A	B26005A	B28001D	B28003A	B29001A
B2A003A	B2A003B	B2A003C	B33301A	B35101A	B37106A	B37301B
B37302A	B38003A	B38003B	B38009A	B38009B	B51001A	B53009A
B54A01C	B54A01H	B55A01A	B61001C	B61001D	B61001F	B61001H
B61001I	B61001M	B61001R	B61001S	B61001W	B67001H	B91001A
B91002A	B91002B	B91002C	B91002D	B91002E	B91002F	B91002G
B91002H	B91002I	B91002J	B91002K	B91002L	B95030A	B95061A
B95061F	B95061G	B95077A	B97103E	B97104G	BA1101B	BC1109A
BC1109C	BC1109D	BC1202A	BC1202B	BC1202E	BC1202F	BC1202G
BC2001D	BC2001E					

The following 9 tests contain modifications to their respective source code files:

AD7006A wrongly assumes that an expression in an assignment statement is of type universal integer, and so should deliver a

correct result that is in the range of type INTEGER. This implementation is correct in treating the expression a being of type INTEGER; an exception is raised because the operand SYSTEM.MEMORY_SIZE exceeds INTEGER'LAST.

The implementer's modification of this test (declaring the assigned -- variable I to be of type LONG_INTEGER) is ruled to be an acceptable means to passing this test by the AVO.

C34007A, C34007D, C34007G, C34007J, C34007M, C34007P, C34007S, and C87B62B (8 tests) The AVO accepts the implementer's argument that, without there being a STORAGE_SIZE length clause for an access type, the meaning of the attribute 'STORAGE_SIZE is undefined for that type. Therefore, a length clause has been added in these tests in order to alter the default size of a collection. 1024 was used for all of the above tests except for C34007D and C34007G which used 2048.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the DACS-386/UNIX, Version 4.4 compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the DACS-386/UNIX, Version 4.4 compiler using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host computer:	RC900
Host operating system:	UNIX V, Release 1.0.4
Target computer:	RC900
Target operating system:	UNIX V, Release 1.0.4
Compiler:	DACS-386/UNIX, Version 4.4

The ACVC Test Suite was loaded onto a VAX-8350 from the magnetic tape. The ACVC Test Suite was then downloaded onto an ICL DRS330 from the VAX-8530 via Ethernet (using DNICP net software utility) via SUN-3/60 Workstation via streamer tape to the ICL DRS300.

A magnetic tape containing all tests except for withdrawn tests was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized on-site. Tests requiring modifications during the prevalidation testing were modified on-site.

TEST INFORMATION

The contents of the magnetic tape were loaded onto a VAX-8350 and transferred to the host computer, ICL DRS300, via Ethernet (using DNICP net software utility) via SUN-3/60 Workstation via streamer tape.

After the test files were loaded to disk, the full set of tests was compiled and linked on the RC900, and all executable tests were run on the RC900. Results were transferred using the following procedure:

From RC900 to VAX-8530:

TWO METHODS were used depending upon availability of the software/hardware: (1) kermit via terminal hardware connection to the VAX-8530; (2) the files on the RC900 are written to a UNIX tar file which is then fragmented into 360KByte groups which are then written to DOS 360 KByte diskettes; the DOS 360 KBytes diskettes are loaded onto an ICL DRS300 where the 360KByte file groups are concatenated to form the original tar file; the original tar file is then copied to a streamer tape which is then loaded onto the Sun-1 Workstation or Sun-3 Workstation; from the Sun-1 or the Sun-3 the file(s) are transferred by Ethernet (using DNICP net software utility) to the VAX-8530.

The compiler was tested using command scripts provided by DDC INTERNATIONAL A/S and reviewed by the validation team. The compiler was tested using the following option settings. See Appendix E for a complete listing of the compiler options for this implementation.

```
-L
-/s
```

Tests were compiled, linked, and executed (as appropriate) using a single host and target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. Selected listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at Lyngby, Denmark and was completed on 27 October 1989.

APPENDIX A

DECLARATION OF CONFORMANCE

DDC INTERNATIONAL A/S has submitted the following Declaration of Conformance concerning the DACS-386/UNIX, Version 4.4.

DECLARATION OF CONFORMANCE

Compiler Implementor:	DDC International A/S Gl. Lundtoftevej 1B 2800 Lyngby, Denmark
Ada Validation Facility:	Ada Validation Facility National Computer Systems Laboratory (NCSL) National Institute of Standards and Technology Building 225, Room A266 Gaitherburg, MD 20899, U.S.A.

Ada Compiler Validation Capability (ACVC) Version: 1.10

Base Configuration

Base Compiler Name:	DACS-386/UNIX, Version 4.4
Host Architecture:	RC900
Host OS and Version:	UNIX V, Release 1.0.4
Target Architecture:	Same as host
Target OS and Version:	Same as host

Implementor's Declaration

I, the undersigned, representing DDC International A/S, have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that DDC International A/S is the owner of record of the Ada language compiler(s) listed above, and as such, is responsible for maintaining said compiler(s) in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.

Date: 25 October 1989

DDC International A/S
Hasse Hansson, Department Manager

Owner's Declaration

I, the undersigned, representing DDC International A/S, take full responsibility for implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. I declare that all of the Ada language compilers listed, and their host/taget performance, are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

Date: 25 October 1989

DDC International A/S
Hasse Hansson, Department Manager

DDC International A/S | Gl Lundtoftevej 1B
DK-2800 Lyngby
Denmark | Telephone:
+45 | Telex: | Telefax:
+45

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the DACS-386/UNIX, Version 4.4 compiler, as described in this Appendix, are provided by DDC INTERNATIONAL A/S. Unless specifically noted otherwise, references in this appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

...

 type SHORT_INTEGER is range -32_768 .. 32_767;
 type INTEGER is range -2_147_483_648 .. 2_147_483_647;
 type LONG_INTEGER is range -2**63 .. 2**63 - 1;

 type FLOAT is digits 6 range
            -3.40282366920938E+38 .. 1.93428038904620E+38;
 type LONG_FLOAT is digits 15 range
            -1.7976931348623157E+308 .. 1.7976931348623157E+308;

 type DURATION is delta 2**(-14) range
            -131_072.00000 .. 131_071.00000 ;

...

end STANDARD;
```

F Implementation-Dependent Characteristics

This appendix describes the implementation-dependent characteristics of DACS-386®/UNIX® as required in Appendix F of the Ada Reference Manual (ANSI/MIL-STD-1815A).

F.1 Implementation-Defined Pragmas

This section describes all implementation-defined pragmas.

F.1.1 Pragma INTERFACE_SPELLING

This pragma allows an Ada program to call a non-Ada program whose name contains characters that would be an invalid Ada subprogram identifier. It can also be used when subprogram names are case sensitive, e.g. C routines. This pragma must be used in conjunction with pragma INTERFACE, i.e. pragma INTERFACE must be specified for the non-Ada subprogram name prior to using pragma INTERFACE_SPELLING.

The pragma has the format:

```
pragma INTERFACE_SPELLING (subprogram name, string literal);
```

where the subprogram name is that of one previously given in pragma INTERFACE and the string literal is the exact spelling of the interfaced subprogram in its native language.

F.1.2 Pragma INTERRUPT_HANDLER

The DACS-386®/UNIX® allows the use of pragma INTERRUPT_HANDLER for compatibility with other DACS compiler systems. In this implementation this pragma does not have any affect in the Ada program. The following information is reference only. It has the format:

```
pragma INTERRUPT_HANDLER;
```

The pragma must appear as the first thing in the specification of the task object. The task must be specified in a package and not a procedure.

F.1.3 Pragma LT_STACK_SPACE

DACS-386®/UNIX® allows use of the pragma LT_Stack_Space for compatibility with other DACS compiler systems. In this implementation, the pragma does not have any effect on the Ada program. The following information is for reference only.

This pragma sets the size of a library task stack segment. The pragma has the format:

```
pragma  LT_STACK_SPACE (T, N);
```

where T denotes either a task object or task type and N designates the size of the library task stack segment in words.

The size of the library task stack is normally specified via the representation clause:

```
for  T'STORAGE_SIZE  use N;
```

The size of the library task stack segment determines how many tasks that can be created which are nested within the library task. All tasks created within a library task will have their stacks allocated from the same segment as the library task stack. Thus, pragma LT_STACK_SPACE must be specified to reserve space within the library task stack segment so that nested tasks' stacks may be allocated.

The following restrictions are placed on the use of LT_STACK_SPACE:

1) It must be used only for library tasks.

2) It must be placed immediately after the task object or type name declaration.

3) The library task stack segment size (N) must be greater than or equal to the library task stack size.

F.2 Implementation-Dependent Attributes

No implementation-dependent attributes are defined.

F.3 Package System

The package System for DACS-386®/UNIX® is:

```
package System is

        type      Word                    is new Short_Integer;
        type      DWord                   is new Integer;
        type      QWord                   is new Long_Integer
        type      UnsignedWord            is range 0..65535;
        for       UnsignedWord'SIZE       use 16;
        type      UnsignedDword           is range 0..16#FFFF_FFFF#;
        for       UnsignedDWord'SIZE      use 32;
        type      Byte                    is range 0..255;
        for       Byte'SIZE               use 8;
        subtype   SegmentId               is UnsignedWord;
        type      Address is record
                  offset                  : UnsignedDWord;
        end record;

        subtype Priority                  is integer range 0..31;
        type    Name                      is (iapx386_FM);

        System_Name   :  constant Name :=              iapx386_FM,
        Storage_Unit  :  constant      :=                    16;
        Memory_Size   :  constant      :=        16#1_0000_0000#;
        Min_Int       :  constant      := -16#8000_0000_0000_0000#;
        Max_Int       :  constant      :=  16#7FFF_FFFF_FFFF_FFFF#;
        Max_Digits    :  constant      :=                    15;
        Max_Mantissa  :  constant      :=                    31;
        Fine_Delta    :  constant      :=          2#1.0#E-31;
        Tick          :  constant      :=     0.000_000_062_5;
                                          --machine dependent

        type   Interface_Language is (ASM86,    PLM86,    C86,
                                      C86_REVERSE,    ASM_ACF,
                                      PLM_ACF,          C_ACF,
                                      C_RESERVE_ACF, ASM_NOACF,
                                      PLM_NOACF,      C_NOACF,
                                      C_REVERSE_NOACF);
        type   ExceptionId is record
             unit_number    : UnsignedDWord;
             unique_number  : UnsignedDWord;
        end record;

        type    TaskValue     is new Integer;
        type    AccTaskValue  is access TaskValue;
```

```
        type    Semaphore       is record
             counter            :  integer;
             first              :  TaskValue;
             last               :  TaskValue;
        end record;

        InitSemaphore           : constant Semaphore'(1, 0, 0);

end SYSTEM;
```

F.4 Representation Clauses

F.4.1 Length Clause

A size attribute for a type T is accepted in the following cases:

- If T is a discrete type then the specified size must be greater than ~~an~~ or equal to the number of bits needed to represent a value of the type, and less than or equal to 32.

- If T is a fixed point type, a floating point type, an access type or a task type the specified size must be equal to the number of bits used to represent values of the type.

- If T is a record type that is not derived then the specified size must be greater than or equal to the number of bits used to represent value of the type.

- If T is an array type that is not derived, and has a size known at compile time then the specified size must be equal to the number of bits used to represent values of the type. In all other cases the size attribute is not accepted.

Furthermore, the size attribute has only effect if the type is part of a composite type.

- Using the STORAGE_SIZE attribute for a collection will set an upper limit on the total size of objects allocated in this collection. If further allocation is attempted, the exception STORAGE_ERROR is raised.

- When STORAGE_SIZE is specified in a length clause for a task, the process stack area will be of the specified size. The process stack area will be allocated inside the "standard" stack segment.

F.4.2 Enumeration Representation Clause

Enumeration representation clauses may specify representations in the range of INTEGER'FIRST + 1..INTEGER'LAST - 1.

Enumeration representation clauses are not supported for derived types.

F.4.3 Record Representation Clauses

When representation clauses are applied to records the following restrictions are imposed:

- the component type is a discrete type different from LONG_INTEGER,
- the component type is an array with a discrete element type different from LONG_INTEGER,
- if the component is a record or an unpacked array, it must start on a storage unit boundary, a storage unit being 16 bits,
- a record occupies an integral number of storage units,
- a record may take up a maximum of 32K storage units,
- a component must be specified with its proper size (in bits), regardless of whether the component is an array or not,
- if a non-array component has a size which equals or exceeds one storage unit (16 bits), the component must start on a storage unit boundary, i.e. the component must be specified as:

  component at N range 0..16 * M - 1;

  where N specifies the relative storage unit number (0,1,...) from the beginning of the record, and M the required number of storage units (1,2,...)
- the elements in an array component should always be wholly contained in one storage unit,
- if a component has a size which is less than one storage unit, it must be wholly contained within a single storage unit:

  component at N range X .. Y;

  where N is as in the previous paragraph, and 0 <= X <= Y <= 15

If the record type contains components which are not covered by a component clause, they are allocated consecutively after the component with the value. Allocation of a record component without a component clause is always aligned on a storage unit boundary. Holes created because of component clauses are not otherwise utilized by the compiler.

F.4.3.1 Alignment Clauses

Alignment clauses for records are implemented with the following characteristics:

- If the declaration of the record type is done at the outermost level in a library package, any alignment is accepted.
- If the record declaration is done at a given static level (higher than the outermost library level, i.e. the permanent area), only word alignments are accepted.
- Any record object declared at the outermost level in a library package will be aligned according to the alignment clause specified for the type. Record objects declared elsewhere can only be aligned on a word boundary. If the record type has been associated a different alignment, an error message will be issued.
- If a record type with an associated alignment clause is used in a composite type, the alignment is required to be one word: an error message is issued if this is not the case.

F.5 Implementation-Dependent Names for Implementation-Dependent Coponents

None defined by the compiler.

F.6 Address Clauses

In the Dacs-386®/UNIX® implementation only static address clauses are allowed, i.e. either a literal or a static expression.

F.7 Unchecked Conversions

Unchecked conversion is only allowed between objects of the same size.

F.8 Input/Output Packages

The implementation supports all requirements of the Ada language. It is an effective interface to the UNIX file system, and in the case of the text I/O also an effective interface to the UNIX standard input, standard output, and standard error streams.

This section describes the functional aspects of the interface to the UNIX file system, including the methods of using the interface to take advantage of the file control facilities provided.

The Ada input-output concept as defined in Chapter 14 of the ARM does not constitute a complete functional specification of the input-output packages. Some aspects of the I/O system are not described at all, with others intentionally left open for implementation. This section describes those sections not covered in the ARM.

The UNIX operating system considers all files to be sequences of characters. Files can either be accessed sequentially or randomly. Files are not structured into records, but an access routine can treat a file as a sequence of records if it arranges the record level input-output. Two restrictions that apply are:

- If a direct access (using lseek(2)) to standard input, standard output, or standard error will cause a USE_ERROR to occur.

- Attempting to direct access (using lseek(2), open(2), mknod(2), or pipe(2)) a UNIX pipe or FIFO will cause a USE_ERROR to occur.

Note that for sequential or text files (Ada files not UNIX external files) RESET on a file in mode OUT_FILE will empty the file. Also, a sequential or text file opened as an OUT_FILE will be emptied.

F.8.1 External Files

An external file is either a UNIX disk file, a UNIX FIFO, a UNIX pipe, or any device defined in the UNIX directory. The use of devices such as a tape drive or communication line may require special access permissions or have restrictions. If an inappropriate operation is attempted on a device, the USE_ERROR exception is raised.

External files created within the UNIX file system shall exist after the termination of the program that created it, and will be accessible from other Ada programs. A form created with the FORM parameter will also exist after program termination. However, pipes and temporary files will not exist after program termination.

Creation of a file with the same name as an existing external file will cause the existing file to be overwritten.

Creation of files with mode IN_FILE will cause USE_ERROR to be raised.

The name parameter to the input/output routines must be a valid UNIX file name. If the name parameter is empty, then a temporary file is created in the /tmp directory. This file is automatically deleted when the program that created it terminates.

F.8.2 File Management

This section provides useful information for performing file management functions within an Ada program.

The only restrictions in performing Sequential and Direct I/O are:

- The maximum size of an object of ELEMENT_TYPE is 2_147_483_647 bits.
- If the size of an object of ELEMENT_TYPE is variable, the maximum size must be determinable at the point of instantiation from the value of the SIZE attribute.

The NAME parameter

The NAME parameter must be a valid UNIX pathname (unless null). If any directory in the pathname is inaccessible, a USE_ERROR is raised.

The UNIX names "stdin", "stdout", and "stderr" can be used with TEXT_IO.OPEN. No physical opening of the external file is performed and the Ada file will be associated with the already open external file. These names have no significance for other packages.

Temporary files (NAME = null string) are created using tmpname(3) and are deleted when CLOSED. Abnormal program termination may leave temporary files in existence.

The FORM parameter

The FORM parameter has the following facilities:

- A FIFO special file can be opened using the open(2) system call. This is achieved with the "FIFO" string. Note that this cannot be used with CREATE.

  The default value for this facility is "ORDINARY", which designates the creation of an ordinary file.

An additional flag associated with FIFO specials is provided to allow waiting or immediate return. This flag, and its status, is specified with the additional strings, "O_NDELAY=ON" for ON and "O_NDELAY=OFF" for OFF. Default is "O_NDELAY=OFF".

- The "APPEND" string can be used to open text files without emptying the file. This parameter cannot be used with CREATE. The default condition is "NOAPPEND".

- Access rights to a file can be controlled by using a "MODE=<mode>" string in the CREATE procedure. <mode> is an octal, decimal, or hexadecimal integer in the standard UNIX format. Default mode is 0644. This facility can also be used by OPEN to change access permissions on existing files by means of the chmod(2) system call.

If more than one of the three options (FIFO, APPEND, and MODE) are included, the rightmost option is selected. Blanks are not significant within any part of the string. The syntax of the FORM parameter provides all default options as required in the Ada Reference Manual:

<form_parameter> := [<option> ,...]

where <option> := <access rights> | <fifo> | <append>

<access_rights> is MODE=<mode #>. The mode number can be in decimal, octal (0###), or hexadecimal (0##).

<fifo> is either "FIFO [O_NDELAY= ON|OFF]" or "ORDINARY"

<append> is either "APPEND" or "NOAPPEND"

File Access

The following guidelines should be observed when performing file I/O operations:

- At a given instant, any number of files in an Ada program can be associated with corresponding external files.

- When sharing files between programs, it is the responsibility of the programmer to determine the effects of sharing files.

- The RESET and OPEN operations to files of mode OUT_FILE will empty the contents of the file in SEQUENTIAL_IO and TEXT_IO.

- Files can be interchanged between SEQUENTIAL_IO and DIRECT_IO without any special operations if the files are of the same object type.

F.8.3 Package TEXT_IO

The specification of package TEXT_IO:

```
with BASIC_IO_TYPES;

with IO_EXCEPTIONS;
package TEXT_IO is

   type FILE_TYPE is limited private;

   type FILE_MODE is (IN_FILE, OUT_FILE);

   type  COUNT is range 0 .. LONG_INTEGER'LAST;
   subtype POSITIVE_COUNT is COUNT range 1 .. COUNT'LAST;
   UNBOUNDED: constant COUNT:= 0; -- line and page length

   -- max. size of an integer output field 2#....#
   subtype FIELD is INTEGER range 0 .. 35;

   subtype NUMBER_BASE is INTEGER range 2 .. 16;

   type TYPE_SET is (LOWER_CASE, UPPER_CASE);

   -- File Management

   procedure CREATE (FILE  : in out FILE_TYPE;
                     MODE  : in FILE_MODE :=OUT_FILE;
                     NAME  : in STRING    :="";
                     FORM  : in STRING    :=""
                     );

   procedure OPEN   (FILE  : in out FILE_TYPE;
                     MODE  : in FILE_MODE;
                     NAME  : in STRING;
                     FORM  : in STRING    :=""
                     );

   procedure CLOSE   (FILE : in out FILE_TYPE)
   procedure DELETE  (FILE : in out FILE_TYPE);
   procedure RESET   (FILE : in out FILE_TYPE;  MODE: in FILE_MODE);
   procedure RESET   (FILE : in out FILE_TYPE);

   function MODE     (FILE : in FILE_TYPE) return FILE_MODE;
   function NAME     (FILE : in FILE_TYPE) return STRING;
   function FORM     (FILE : in FILE_TYPE) return STRING;

   function IS_OPEN  (FILE : in FILE_TYPE return BOOLEAN;

    -- control of default input and output files
```

```
procedure SET_INPUT  (FILE : in FILE_TYPE);
procedure SET_OUTPUT (FILE : in FILE_TYPE);

function STANDARD_INPUT   return FILE_TYPE;
function STANDARD_OUTPUT  return FILE_TYPE;

function CURRENT_INPUT    return FILE_TYPE;
function CURRENT_OUTPUT   return FILE_TYPE;

-- specification of line and page lengths

procedure SET_LINE_LENGTH  (FILE : in FILE_TYPE; TO : in COUNT);
procedure SET_LINE_LENGTH  (TO : in COUNT);

procedure SET_PAGE_LENGTH  (FILE : in FILE_TYPE; TO : in COUNT);
procedure SET_PAGE_LENGTH  (TO : in COUNT);

function  LINE_LENGTH      (FILE : in FILE_TYPE) return COUNT;
function  LINE_LENGTH      return COUNT;

function  PAGE_LENGTH      (FILE : in FILE_TYPE) return COUNT;
function  PAGE_LENGTH      return COUNT;

-- Column, Line, and Page Control

procedure NEW_LINE   (FILE    : in FILE_TYPE;
                      SPACING : in POSITIVE_COUNT := 1);
procedure NEW_LINE   (SPACING : in POSITIVE_COUNT := 1);

procedure SKIP_LINE  (FILE    : in FILE_TYPE;
                      SPACING : in POSITIVE_COUNT := 1);
procedure SKIP_LINE  (SPACING : in POSITIVE_COUNT := 1);

function END_OF_LINE (FILE : in FILE_TYPE) return BOOLEAN;
function  END_OF_LINE                      return BOOLEAN;

procedure NEW_PAGE   (FILE : in FILE_TYPE);
procedure NEW_PAGE;

procedure SKIP_PAGE  (FILE : in FILE_TYPE);
procedure SKIP_PAGE;

function END_OF_PAGE (FILE : in FILE_TYPE) return BOOLEAN;
function  END_OF_PAGE                      return BOOLEAN;

function END_OF_FILE (FILE : in FILE_TYPE) return BOOLEAN;
function  END_OF_FILE                      return BOOLEAN;

procedure SET_COL    (FILE : in FILE_TYPE;TO :in POSITIVE_COUNT);
procedure SET_COL    (TO   : in POSITIVE_COUNT);

procedure SET_LINE   (FILE : in FILE_TYPE;TO :in POSITIVE_COUNT);
```

```
procedure SET_LINE   (TO   : in POSITIVE_COUNT);

function  COL        (FILE : in FILE_TYPE) return POSITIVE_COUNT;
function  COL                              return POSITIVE_COUNT;

function  LINE       (FILE : in FILE_TYPE) return POSITIVE_COUNT;
function  LINE                             return POSITIVE_COUNT;

function  PAGE       (FILE : in FILE_TYPE) return POSITIVE_COUNT;
function  PAGE                             return POSITIVE_COUNT;

-- Character Input-Output

procedure GET        (FILE : in FILE_TYPE; ITEM : out CHARACTER);
procedure GET        (ITEM : out CHARACTER);
procedure PUT        (FILE : in FILE_TYPE; ITEM : in CHARACTER);
procedure PUT        (ITEM : in CHARACTER);

-- String Input-Output

procedure GET        (FILE : in FILE_TYPE; ITEM : out CHARACTER);
procedure GET        (ITEM : out CHARACTER);
procedure PUT        (FILE : in FILE_TYPE; ITEM : in CHARACTER);
procedure PUT        (ITEM : in CHARACTER);

procedure GET_LINE   (FILE : in FILE_TYPE;
                      ITEM : out STRING;
                      LAST : out NATURAL);

procedure GET_LINE   (ITEM : out STRING;    LAST : out NATURAL);
procedure PUT_LINE   (FILE : in FILE_TYPE;  ITEM : in STRING);
procedure PUT_LINE   (ITEM : in STRING);
```

```
-- Generic Package for Input-Output of Integer Types

generic
    type NUM is range <>; is

   DEFAULT_WIDTH : FIELD       := NUM'WIDTH;
   DEFAULT_BASE  : NUMBER_BASE :=          10;

   procedure GET  (FILE        : in FILE_TYPE;
                   ITEM        : out NUM;
                   WIDTH       : in FIELD := 0);

   procedure GET  (ITEM        : out NUM;
                   WIDTH       : in FIELD := 0);

   procedure PUT  (FILE        : in FILE_TYPE;
                   ITEM        : in NUM;
                   WIDTH       : in FIELD       := DEFAULT_WIDTH;
                   BASE        : in NUMBER_BASE := DEFAULT_BASE);

   procedure PUT  (ITEM        : in NUM;
                   WIDTH       : in FIELD       := DEFAULT_WIDTH;
                   BASE        : in NUMBER_BASE := DEFAULT_BASE);

   procedure GET  (FROM        : in STRING;
                   ITEM        : out NUM;
                   LAST        : out POSITIVE);

   procedure PUT  (TO          : out STRING;
                   ITEM        : in NUM;
                   BASE        : in NUMBER_BASE := DEFAULT_BASE);

end INTEGER_IO;
```

```
   -- Generic Packages for Input-Output of Real Types

generic
   type NUM is digits <>;
package FLOAT_IO; is

   DEFAULT_FORE : FIELD :=              2;
   DEFAULT_AFT  : FIELD := NUM'DIGITS - 1;
   DEFAULT_EXP  : FIELD :=              3;

   procedure GET  (FILE     : in FILE_TYPE;
                   ITEM     : out NUM;
                   WIDTH    : in FIELD := 0);
   procedure GET  (ITEM     : out NUM;
                   WIDTH    : in FIELD := 0);

   procedure PUT  (FILE     : in FILE_TYPE;
                   ITEM     : in NUM;
                   FORE     : in FIELD := DEFAULT_FORE;
                   AFT      : in FIELD := DEFAULT_AFT;
                   EXP      : in FIELD := DEFAULT_EXP);
   procedure PUT  (ITEM     : in NUM;
                   FORE     : in FIELD := DEFAULT_FORE;
                   AFT      : in FIELD := DEFAULT_AFT;
                   EXP      : in FIELD := DEFAULT_EXP);

   procedure GET  (FROM     : in STRING;
                   ITEM     : out NUM;
                   LAST     : out POSITIVE);
   procedure PUT  (TO       : out STRING;
                   ITEM     :  in NUM;
                   AFT      :  in FIELD    := DEFAULT_AFT;
                   EXP      :  in FIELD    := DEFAULT_EXP);

end FLOAT_IO;
```

```
generic
   type NUM is delta <>;
package FIXED_IO is

   DEFAULT_FORE    : FIELD := NUM'FORE;
   DEFAULT_AFT     : FIELD := NUM'AFT;
   DEFAULT_EXP     : FIELD := 0;

   procedure GET  (FILE    : in FILE_TYPE;
                   ITEM    : out NUM;
                   WIDTH   : in FIELD     := 0);
   procedure GET  (ITEM    : out NUM;
                   WIDTH   : in FIELD     := 0);

   procedure PUT  (FILE    : in FILE_TYPE;
                   ITEM    : in NUM;
                   FORE    : in FIELD := DEFAULT_FORE;
                   AFT     : in FIELD := DEFAULT_AFT;
                   EXP     : in FIELD := DEFAULT_EXP);
   procedure PUT  (ITEM    : in NUM;
                   FORE    : in FIELD := DEFAULT_FORE;
                   AFT     : in FIELD := DEFAULT_AFT;
                   EXP     : in FIELD := DEFAULT_EXP);

   procedure GET  (FROM    : in STRING;
                   ITEM    : out NUM;
                   LAST    : out POSITIVE);

   procedure PUT  (TO      : out STRING;
                   ITEM    :  in NUM;
                   AFT     :  in FIELD := DEFAULT_AFT;
                   EXP     :  in FIELD := DEFAULT_EXP);

end FIXED_IO;
```

```
   -- Generic Package for Input-Output of Enumeration Types

generic
   type ENUM is (<>);
package ENUMERATION_IO is

   DEFAULT_WIDTH            : FIELD         := 0;
   DEFAULT_SETTING          : TYPE_SET      := UPPER_CASE;

   procedure GET  (FILE : in FILE_TYPE; ITEM : out ENUM);
   procedure GET  (ITEM : out ENUM);

   procedure PUT  (FILE  : FILE_TYPE;
                   ITEM  : in ENUM;
                   WIDTH : in FIELD        := DEFAULT_WIDTH;
                   SET   : in TYPE_SET     := DEFAULT_SETTING);
   procedure PUT  (ITEM  : in ENUM;
                   WIDTH : in FIELD        := DEFAULT_WIDTH;
                   SET   : in TYPE_SET     := DEFAULT_SETTING);

   procedure GET  (FROM  : in  STRING;
                   ITEM  : out ENUM;
                   LAST  : out POSITIVE);

   procedure PUT  (TO    : out STRING;
                   ITEM  : in  ENUM;
                   SET   : in  TYPE_SET    := DEFAULT_SETTING);

end ENUMERATION_IO;


   -- Exceptions

   STATUS_ERROR;    : exception renames IO_EXCEPTIONS.STATUS_ERROR;
   MODE_ERROR;      : exception renames IO_EXCEPTIONS.MODE_ERROR;
   NAME_ERROR;      : exception renames IO_EXCEPTIONS.NAME_ERROR;
   USE_ERROR;       : exception renames IO_EXCEPTIONS.USE_ERROR;
   DEVICE_ERROR;    : exception renames IO_EXCEPTIONS.DEVICE_ERROR;
   END_ERROR;       : exception renames IO_EXCEPTIONS.END_ERROR;
   DATA_ERROR;      : exception renames IO_EXCEPTIONS.DATA_ERROR;
   LAYOUT_ERROR;    : exception renames IO_EXCEPTIONS.LAYOUT_ERROR;

private

   type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

end TEXT_IO;
```

F.8.4 Package LOW_LEVEL_IO

The specification of LOW_LEVEL_IO is:

```
with System;

package LOW_LEVEL_IO is

  subtype port_address is System.UnsignedWord;

  type ll_io_8      is new short_integer range -128..127;
  type ll_io_16     is new short_integer;
  type ll_io_32     is new integer;

  procedure send_control(device : in port_address;
                         data   : in System.Byte);
   -- unsigned 8 bit entity

  procedure send_control(device : in port_address;
                         data   : in System.UnsignedWord);
   -- unsigned 16 bit entity

  procedure send_control(device : in port_address;
                         data   : in System.UnsignedDWord);
   -- unsigned 32 bit entity

  procedure send_control(device : in port_address;
                         data   : in ll_io_8);
   -- signed  8 bit entity

  procedure send_control(device : in port_address;
                         data   : in ll_io_16);
   -- signed 16 bit entity

  procedure send_control(device : in port_address;
                         data   : in ll_io_32);
   -- signed 32 bit entity

  procedure receive_control(device : in port_address;
                            data   : out System.Byte);
   -- unsigned  8 bit entity

  procedure receive_control(device : in port_address;
                            data   : out System.UnsignedWord);
   -- unsigned 16 bit entity

  procedure receive_control(device : in port_address;
                            data   : out System.UnsignedDWord);
   -- unsigned 32 bit entity

  procedure receive_control(device : in port_address;
                            data   : out ll_io_8);
```

```
   -- signed  8 bit entity

  procedure receive_control(device : in port_address;
                            data   : out ll_io_16);
   -- signed 16 bit entity

  procedure receive_control(device : in port_address;
                            data   : out ll_io_32);
   -- signed 32 bit entity


  private

     pragma inline(send_control, receive_control);

end LOW_LEVEL_IO;
```

F.8.5 Package SEQUENTIAL_IO

In SEQUENTIAL_IO, type checking for DATA_ERROR has been excluded for elements of an unconstrained type.

```
-- Source code for SEQUENTIAL_IO

with IO_EXCEPTIONS;

generic

   type ELEMENT_TYPE is private;

package SEQUENTIAL_IO is

  type FILE_TYPE is limited private;

  type FILE_MODE is (IN_FILE, OUT_FILE);

-- File management


   procedure CREATE(FILE   : in out FILE_TYPE;
                    MODE   : in     FILE_MODE  := OUT_FILE;
                    NAME   : in     STRING     := "";
                    FORM   : in     STRING     := "");

   procedure OPEN  (FILE   : in out FILE_TYPE;
                    MODE   : in     FILE_MODE;
                    NAME   : in     STRING;
                    FORM   : in     STRING  := "");

   procedure CLOSE (FILE   : in out FILE_TYPE);
```

```
   procedure DELETE(FILE    : in out FILE_TYPE);

   procedure RESET (FILE    : in out FILE_TYPE; MODE : in FILE_MODE);

   procedure RESET (FILE    : in out FILE_TYPE);

   function MODE   (FILE    : in FILE_TYPE) return FILE_MODE;

   function NAME   (FILE    : in FILE_TYPE) return STRING;

   function FORM   (FILE    : in FILE_TYPE) return STRING;

   function IS_OPEN(FILE    : in FILE_TYPE) return BOOLEAN;

-- input and output operations

   procedure READ  (FILE    : in FILE_TYPE;
                    ITEM    : out ELEMENT_TYPE);

   procedure WRITE (FILE    : in FILE_TYPE;
                    ITEM    : in ELEMENT_TYPE);

   function END_OF_FILE
                   (FILE    : in FILE_TYPE) return BOOLEAN;

-- exceptions

   STATUS_ERROR    : exception renames IO_EXCEPTIONS.STATUS_ERROR;
   MODE_ERROR      : exception renames IO_EXCEPTIONS.MODE_ERROR;
   NAME_ERROR      : exception renames IO_EXCEPTIONS.NAME_ERROR;
   USE_ERROR       : exception renames IO_EXCEPTIONS.USE_ERROR;
   DEVICE_ERROR    : exception renames IO_EXCEPTIONS.DEVICE_ERROR;
   END_ERROR       : exception renames IO_EXCEPTIONS.END_ERROR;
   DATA_ERROR      : exception renames IO_EXCEPTIONS.DATA_ERROR;

private

   type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

end SEQUENTIAL_IO;
```

F.8.6 Package DIRECT_IO

In DIRECT_IO, type checking for DATA_ERROR has been excluded for elements of an unconstrained type.

```
with BASIC_IO_TYPES;
with IO_EXCEPTIONS;

generic

        type ELEMENT_TYPE is private;

package DIRECT_IO is

        type FILE_TYPE is limited private;

        type FILE_MODE is (IN_FILE, INOUT_FILE, OUT_FILE);

        type COUNT is range 0..LONG_INTEGER'LAST;

        subtype POSITIVE_COUNT is COUNT range 1..COUNT'LAST;

    -- File management

  procedure CREATE(FILE    : in out FILE_TYPE;
                   MODE    : in     FILE_MODE  := INOUT_FILE;
                   NAME    : in     STRING     := "";
                   FORM    : in     STRING     := "");

   procedure OPEN  (FILE    : in out FILE_TYPE;
                    MODE    : in       FILE_MODE;
                    NAME    : in       STRING;
                    FORM    : in       STRING  := "");

   procedure CLOSE (FILE    : in out FILE_TYPE);

   procedure DELETE(FILE    : in out FILE_TYPE);

   procedure RESET (FILE    : in out FILE_TYPE;
                    MODE    : in       FILE_MODE);

   procedure RESET (FILE    : in out FILE_TYPE);

   function MODE   (FILE    : in    FILE_TYPE) return FILE_MODE;

   function NAME   (FILE    : in    FILE_TYPE) return STRING;

   function FORM   (FILE    : in    FILE_TYPE) return STRING;

   function IS_OPEN(FILE    : in    FILE_TYPE) return BOOLEAN;
```

```
-- input and output operations

   procedure READ  (FILE   : in    FILE_TYPE;
                    ITEM   : out   ELEMENT_TYPE;
                    FROM   : in    POSITIVE_COUNT);
   procedure READ  (FILE   : in    FILE_TYPE;
                    ITEM   : out   ELEMENT_TYPE);

   procedure WRITE (FILE   : in    FILE_TYPE;
                    ITEM   : in    ELEMENT_TYPE;
                    TO     : in    POSITIVE_COUNT);
   procedure WRITE (FILE   : in    FILE_TYPE;
                    ITEM   : in    ELEMENT_TYPE);

   procedure SET_INDEX
                   (FILE   : in FILE_TYPE;
                    TO     : in POSITIVE_COUNT);

   function INDEX  (FILE   : in FILE_TYPE) return POSITIVE_COUNT;

   function SIZE   (FILE   : in FILE_TYPE) return COUNT;

   function END_OF_FILE
                   (FILE   : in FILE_TYPE) return BOOLEAN;

-- exceptions

   STATUS_ERROR    : exception renames IO_EXCEPTIONS.STATUS_ERROR;
   MODE_ERROR      : exception renames IO_EXCEPTIONS.MODE_ERROR;
   NAME_ERROR      : exception renames IO_EXCEPTIONS.NAME_ERROR;
   USE_ERROR       : exception renames IO_EXCEPTIONS.USE_ERROR;
   DEVICE_ERROR    : exception renames IO_EXCEPTIONS.DEVICE_ERROR;
   END_ERROR       : exception renames IO_EXCEPTIONS.END_ERROR;
   DATA_ERROR      : exception renames IO_EXCEPTIONS.DATA_ERROR;

private

   type FILE_TYPE is new BASIC_IO_TYPES.FILE_TYPE;

end DIRECT_IO;
```

F.9 Machine Code Insertions

The reader should be familiar with the code generation strategy and the 80386 instruction set to fully benefit from this section.

As described in chapter 13.8 of the ARM [83] it is possible to write procedures containing only code statements using the predefined package MACHINE_CODE. The package MACHINE_CODE defines the type MACHINE_INSTRUCTION which, used as a record aggregate, defines a machine code insertion. The following sections list the type MACHINE_INSTRUCTION and types on which it depends, give the restrictions, and show an example of how to use the package MACHINE_CODE.

F.9.1 Predefined Types for Machine Code Insertions

The following types are defined for use when making machine code insertions (their type declarations are given in the following pages):

```
type  opcode_type
type  operand_type
type  register_type
type  segment_register
type  machine_instruction
```

The type REGISTER_TYPE defines registers and register combinations. The double register combinations (e.g. BX_SI) can be used only as address operands (BX_SI describing [BX+SI]). The registers STi describe registers on the floating stack. (ST is the top of the floating stack).

The type SEGMENT_REGISTER defines the four segment registers that can be used to overwrite default segments in an address operand.

The type MACHINE_INSTRUCTION is a discriminant record type with which every kind of instruction can be described. Symbolic names may be used in the form

```
name'ADDRESS
```

```
type opcode_type is (
-- 8086 instructions:
                m_AAA,      m_AAD,      m_AAM,      m_AAS,
                m_ADC,      m_ADD,      m_AND,
                m_CALL,     m_CALLn,
                m_CBW,      m_CLC,      m_CLD       m_CLI,
                m_CMC,      m_CMP,      m_CMPS,     m_CWD,
                m_DAA,      m_DAS,
                m_DEC,      m_DIV,      m_HLT,
```

```
                m_IDIV,         m_IMUL,         m_IN,           m_INC,
                m_INT,          m_INTO,         m_IRET,
                m_JA,           m_JAE,          m_JB,           m_JBE,
                m_JC,           m_JCXZ,         m_JE,           m_JG,
                m_JGE,          m_JL,           m_JLE,          m_JNA,
                m_JNAE,         m_JNB,          m_JNBE,         m_JNC
                m_JNE,          m_JNG,          m_JNGE,         m_JNL,
                m_JNLE,         m_JNO,          m_JNP,          m_JNS,
                m_JNZ,          m_JO,           m_JP,           m_JPE,
                m_JPO,          m_JS,           m_JZ,           m_JMP,
                m_LAHF,         m_LDS,          m_LES,          m_LEA,
                m_LOCK,         m_LODS,
                m_LOOP,         m_LOOPE,        m_LOOPNE,       m_LOOPNZ,
                m_LOOPZ,        m_MOV,          m_MOVS,         m_MUL,
                m_NEG,          m_NOP,          m_NOT,          m_OR,
                m_OUT,          m_POP,          m_POPF,         m_PUSH,
                m_PUSHF,
                m_RCL,          m_RCR,          m_ROL,          m_ROR,
                m_REP,          m_REPE,         m_REPNE,
                m_RET,          m_RETP,         m_RETN,         m_RETNP,
                m_SAHF,
                m_SAL,          m_SAR,          m_SHL,          m_SHR,
                m_SBB,          m_SCAS,
                m_STC,          m_STD,          m_STI,          m_STOS,
                m_SUB,          m_TEST,         m_WAIT,         m_XCHG,
                m_XLAT,         m_XOR,

-- 8087/80187/80287 Floating Point Processor instructions
                m_FABS,         m_FADD,         m_FADDD,        m_FADDP,
                m_FBLD,         m_FBSTP,        m_FCHS,         m_FNCLEX,
                m_FCOM,         m_FCOMD,        m_FCOMP,        m_FCOMPD,
                m_FCOMPP,       m_FDECSTP,      m_FDIV,         m_FDIVD,
                m_FDIVP,        m_FDIVR,        m_FDIVRD,       m_FDIVRP,
                m_FFREE,        m_FIADD,        m_FIADDD,       m_FICOM,
                m_FICOMD,       m_FICOMP,       m_FICOMPD,      m_FIDIV,
                m_FIDIVD,       m_FIDIVR,       m_FIDIVRD,
                m_FILD,         m_FILDD,        m_FILDL,        m_FIMUL,
                m_FIMULD,       m_FINCSTP,      m_FNINIT,       m_FIST,
                m_FISTD,        m_FISTP,        m_FISTPD,       m_FISTPL,
                m_FISUB,
                m_FISUBD,       m_FISUBR,       m_FISUBRD,      m_FLD,
                m_FLDD,         m_FLDCW,        m_FLDENV,       m_FLDLG2,
                m_FLDLN2,       m_FLDL2E,       m_FLDL2T,       m_FLDPI,
                m_FLDZ,         m_FLD1,         m_FMUL,         m_FMULD,
                m_FMULP,        m_FNOP,         m_FPATAN,       m_FPREM,
                m_FPTAN,        m_FRNDINT,      m_FRSTOR,       m_FSAVE,
                m_FSCALE,       m_FSETPM,       m_FSQRT,
                m_FST,          m_FSTD,         m_FSTCW,
```

```
                m_FSTENV,       m_FSTP,         m_FSTPD,        m_FSTSW,
                m_FSTSWAX,      m_FSUB,         m_FSUBD,        m_FSUBP,
                m_FSUBR,        m_FSUBRD,       m_FSUBRP,       m_FTST,
                m_FWAIT,        m_FXAM,         m_FXCH,         m_FXTRACT,
                m_FYL2X,        m_FYL2XP1,      m_F2XM1,

-- 80186/80286/80386 instructions:
-- Notice that some immediate versions ot the 8086 instructions
-- only exist on these targets (shifts, rotates, push, imul, ...)

                m_BOUND,        m_CLTS,         m_ENTER,        m_INS,
                m_LAR,          m_LEAVE,        m_LGDT,         m_LIDT
                m_LSL           m_OUTS          m_POPA,         m_PUSHA
                m_SGDT,         m_SIDT,
                m_ARPL,         m_LLDT,         m_LMSW,         m_LTR,
                m_SLDT,         m_SMSW,         m_STR,          m_VERR,
                m_VERW,

-- the 80386 specific instructions:

                m_SETA,         m_SETAE,        m_SETB,         m_SETBE,
                m_SETC,         m_SETE,         m_SETG,         m_SETGE,
                m_SETL,         m_SETLE,        m_SETNA,        m_SETNAE,
                m_SETNB,        m_SETNBE,       m_SETNC,        m_SETNE,
                m_SETNG,        m_SETNGE,       m_SETNL,        m_SETNLE
                m_SETNO,        m_SETNP,        m_SETNS,        m_SETNZ,
                m_SETO,         m_SETP,         m_SETPE,        m_SETPO,
                m_SETS,         m_SETZ,

                m_BSF,          m_BSR,
                m_BT,           m_BTC,          m_BTR,          m_BTS,
                m_LFS,          m_LGS,          m_LSS,
                m_MOVZX         m_MOVSX,
                m_MOVCR,        m_MOVDB,        m_MOVTR,
                m_SHLD,         M_SHRD,

-- the 80387 specific instructions

                m_FUCOM,        m_FUCOMP,       m_FUCOMPP
                m_FPREM1,       m_FSIN,         m_FCOS,         m_FSINCOS
```

```
-- byte/word/dword variants (to be used, when not deductible from
-- context)
                m_ADDCB,        m_ADCW,         m_ADCD,
                m_ADDB,         m_ADDW,         m_ADDD,
                m_ANDB,         m_ANDW,         m_ANDD,
                m_BTW,          m_BTD,
                m_BTCW,         m_BTCD,
                m_BTRW,         m_BTRD,
                m_BTSW,         m_BTSD,
                m_CBWW,         m_CWDE,
                m_CWDW          m_CDQ,
                m_CMPB,         m_CMPW,         m_CMPD,
                m_CMPSB,        m_CMPSW,        m_CMPSD,
                m_DECB,         m_DECW,         m_DECD,
                m_DIVB,         m_DIVW,         m_DIVD,
                m_IDIVB,        m_IDIVW,        m_IDIVD,
                m_IMULB,        m_IMULW,        m_IMULD,
                m_INCB,         m_INCW,         m_INCD,
                m_INSB,         m_INSW,         m_INSD,
                m_LODSB,        m_LODSW,        m_LODSD,
                m_MOVB,         m_MOVW,         m_MOVD,
                m_MOVSB,        m_MOVSW,        m_MOVSD,
                m_MOVSXB,       m_MOVSXW,
                m_MOVZXB,       m_MOVZXW,
                m_MULB,         m_MULW,         m_MULD,
                m_NEGB,         m_NEGW,         m_NEGD,
                m_NOTB,         m_NOTW,         m_NOTD,
                m_ORB,          m_ORW,          m_ORD,
                m_OUTSB,        m_OUTSW,        m_OUTSD,
                m_POPW,         m_POPD,
                m_PUSHW,        m_PUSHD,
                m_RCLB,         m_RCLW,         m_RCLD,
                m_RCRB,         m_RCRW,         m_RCRD,
                m_ROLB,         m_ROLW,         m_ROLD,
                m_RORB,         m_RORW,         m_RORD,
                m_SALB,         m_SALW,         m_SALD,
                m_SARB,         m_SARW,         m_SARD,
                m_SHLB,         m_SHLW,         m_SHLDW,
                m_SHRB,         m_SHRW,         m_SHRDW,
                m_SBBB,         m_SBBW,         m_SBBD,
                m_SCASB,        m_SCASW,        m_SCASD,
                m_STOSB,        m_STOSW,        m_STOSD,
                m_SUBB,         m_SUBW,         m_SUBD,
                m_TESTB,        m_TESTW,        m_TESTD,
                m_XORB,         m_XORW,         m_XORD,
                m_DATAB,        m_DATAW,         m_DATAD

-- Special 'instructions'

                m_label,        m_reset);
```

```
type operand_type is (  none,              -- no operands

   immediate,                              -- 1 immediate operand
   register,                               -- 1 register operand
   address,                                -- 1 address operand
   system_address,                         -- 1 'address operand
   name,                                   -- CALL name
   register_immediate,                     -- 2 operands:  dest is
                                           -- register, source is
                                           -- immediate
   register_register,                      -- 2 register operands
   register_address,                       -- 2 operands: dest
                                           -- is register,
                                           -- source is address
   address_register,                       -- 2 operands: dest is
                                           -- address,
                                           -- source is register
   register_system_address,                -- 2 operands: dest is
                                           -- register,
                                           -- source is 'address
   system_address_register,                -- 2 operands: dest is
                                           -- 'address,
                                           -- source is register
   address_immediate,                      -- 2 operands: dest is
                                           -- 'address,
                                           -- source is immediate
   system_address_immediate,               -- 2 operands: dest is
                                           -- 'address,
                                           -- source is immediate
   immediate_register,                     -- only allowed for OUT port
                                           -- is
                                           -- immediate source is
                                           -- register
   immediate_immediate,                    -- only allowed for ENTER
   register_register_immediate,            -- allowed for
                                           -- IMULimm,SHRDimm, and
                                           -- SHLDimm
   register_address_immediate              -- allowed for IMULimm
   register_system_address_immediate       -- allowed for IMULimm
   address_register_immediate              -- allowed for SHRDimm,
                                           -- SHLDimm
   system_address_register_immediate       -- allowed for SHRDimm,
                                           -- SHLDimm
   );

type register_type is (AX, CX, DX, BX,      -- word registers
                       SP, BP, SI, DI,      --

                       AL, CL, DL, BL,      -- byte registers
                       AH, CH, DH, BH,      --
```

```
                        EAX, ECX, EDX, EBX    -- dword registers
                        ESP, EBP, ESI, EDI

                        ES, CS, SS, DS,       -- selector registers
                        FS, GS

                        BX_SI,  BX_DI,        -- 8086/80186/80286
                        BP_SI,  BP_DI,        -- combinations

                        ST, ST1, ST2, ST3,    -- floating stack
                                              -- registers
                        ST4, ST5, ST6, ST7,
                        nil );

type segment_register is ( ES, CS, SS, DS, FS, GS, nil );

subtype machine_string is string (1..100);

pragma page;
type machine_instruction (operand_kind : operand_type is  record
      opcode   : opcode_type;

      case operand_kind is
         when immediate =>
            immediate                  : integer;

         when register =>
            r_register                 : register_type;

         when address =>
            a_segment                  : register_type;
            a_address_reg              : register_type;
            a_offset                   : integer;

         when system_address =>
            sa_address                 : system.address;

         when name =>
            n_string                   : machine_string;

         when register_immediate =>
            r_i_register               : register_type;
            r_i_immediate              : integer;

         when register_register =>
            r_r_register_to            : register_type;
            r_r_register_from          : register_type;

         when register_address =>
            r_a_register_to            : register_type;
            r_a_segment                : segment_register;
            r_a_address_reg            : register_type;
```

```
   r_a_offset                    : integer;

when address_register =>
   a_r_segment                   : segment_register;
   a_r_address_reg               : register_type;
   a_r_offset                    : integer;
   a_r_register_from             : register_type;

when register_system_address =>
   r_sa_register_to              : register_type;
   r_sa_address                  : system.address

when system_address_register =>
   sa_r_address                  : system.address;
   sa_r_reg_from                 : register_type;

when address_immediate =>
   a_i_segment                   : segment_register;
   a_i_address_reg               : register_type;
   a_i_offset                    : integer;
   a_i_immediate                 : integer;

when system_address_immediate =>
   sa_i_address                  : system.address;
   sa_i_immediate                : integer;

when immediate_register =>
   i_r_register                  : integer;
   i_r_register                  : register_type;

when immediate_immediate =>
   i_i_immediate1                : integer;
   i_i_immediate2                : integer;

when register_register_immediate =>
   r_r_i_register1               : register_type;
   r_r_i_register2               : register_type;
   r_r_i_immediate2              : integer;

when register_address_immediate =>
   r_a_i_register                : register_type;
   r_a_i_segment                 : register_type;
   r_a_i_address_reg             : register_type;
   r_a_i_offset                  : integer;
   r_a_i_immediate               : integer;

when register_system_address_immediate =>
   r_sa_i_register               : register_type;
   addr10                        : system.address;
   r_sa_i_immediate              : integer;
```

```
      when address_register_immediate =>
         a_r_i_register              : register_type;
         a_r_i_segment               : register_type;
         a_r_i_address_reg           : register_type;
         a_r_i_offset                : integer;
         a_r_i_immediate             : integer;

      when system_address_register_immediate =>
         sa_r_i_address              : system.address;
         sa_r_i_register             : register_type;
         sa_r_i_immediate            : integer;

      when others =>
         null;
   end case;
end record;
```

F.9.2 Restrictions

Only procedures, and not functions, may contain machine code insertions. Also procedures that use machine code insertions must be specified with PRAGMA inline.

Symbolic names in the form x'ADDRESS can only be used in the following cases:

1) x is an object of scalar type or access type declared as an object, a formal parameter, or by static renaming.

2) x is an array with static constraints declared as an object (not as a formal parameter or by renaming).

3) x is a record declared as an object (not a formal parameter or by renaming).

All opcodes defined by the type OPCODE_type except the m_CALL can be used.

Two opcodes to handle labels have been defined:

m_label: defines a label. The label number must be in the range 1 <= x <= 25 and is put in the offset field in the first operand of the MACHINE_INSTRUCTION.

m_reset: used to enable use of more than 25 labels. The label number after a m_RESET must be in the range 1 <= x <=25. To avoid errors you must make sure that all used labels have been defined before a reset, since the reset operation clears all used labels.

All floating instructions have at most one operand which can be any of the following:

- a memory address
- a register or an immediate value
- an entry in the floating stack

F.9.3 Examples

The following section contains examples of how to use the machine code insertions and lists the generated code.

F.9.3.1 Example Using Labels

The following assembler code can be described by machine code insertions as shown:

```
     MOV       AX,7
     MOV       CX,4
     CMP       AX,CX
     JG        1
     JE        2
     MOV       CX,AX
  1: ADD       AX,CX
  2: MOV       SS: [BP+DI], AX
```

```
with MACHINE_CODE; use MACHINE_CODE;
package example_MC is

    procedure test_labels;
    pragma inline (test_labels);

end example_MC;

package body example_MC is


procedure test_labels is

begin

  MACHINE_INSTRUCTION'(register_immediate, m_MOV, AX, 7);
  MACHINE_INSTRUCTION'(register_immediate, m_MOV, CX, 4);
  MACHINE_INSTRUCTION'(register_register, m_CMP, AX, CX);
  MACHINE_INSTRUCTION'(immediate, m_JG, 1);
  MACHINE_INSTRUCTION'(immediate, m_JE, 2);
  MACHINE_INSTRUCTION'(register_register, m_MOV, CX, AX);
  MACHINE_INSTRUCTION'(immediate, m_label, 1);
```

```
  MACHINE_INSTRUCTION'(register_register, m_ADD, AX, CX);
  MACHINE_INSTRUCTION'(immediate, m_label, 2);
  MACHINE_INSTRUCTION'(address_register,m_MOV,SS,BP_DI,0,AX);

end test_labels;

end example_MC;
```

F.10 Package Tasktypes

The TaskTypes package defines the TaskControlBlock type.

```
with System;

package TaskTypes is

   subtype Offset    is System.UnsignedDWord;
   subtype BlockId   is System.UnsignedDWord;

   type    TaskEntry     is new System.UnsignedDWord;
   type    EntryIndex    is new System.UnsignedDWord;
   type    AlternativeId is new System.UnsignedDWord;
   type    Ticks         is new System.UnsignedDWord;
   type    Bool          is new Boolean;
   for     Bool'size     use 8;
   type UIntg            is new System.UnsignedDword;

   type Semaphore        is record
                             counter       : Integer;
                             first, last   : System.TaskValue;
                           end record;

   type    TaskState     is (Initial,
                           -- The task is created, but activation
                           -- has not started yet.

                           Engaged,
                           -- The task has called an entry, and the
                           -- call is now accepted, ie. the rendezvous
                           -- is in progress.

                           Running,
                           -- Covers all other states.

                           Delayed,
                           -- The task awaits a timeout to expire.

                            EntryCallingTimed,
                            -- The task has called an entry which
                            -- is not yet accepted.
```

```
                        EntryCallingUnconditional,
                        -- The task has called an entry
                        -- unconditionally,
                        -- which is not yet accepted.

                        SelectingTimed,
                        -- The task is waiting in a select
                        -- statement with an open delay
                        -- alternative.

                        SelectingUnconditional,
                        -- The task waits in a select statement
                        -- entirely with accept statements.

                        SelectingTerminable,
                        -- The task waits in a select statement
                        -- with an open terminate alternative.

                        Accepting,
                        -- The task waits in an accept statement.

                        Synchronizing,
                        -- The task waits in an accept statement
                        -- with no statement list.

                        Completed,
                        -- The task has completed the execution of
                        -- its statement list, but not all
                        -- dependent tasks are terminated.

                        Terminated );
                        -- The task and all its descendants
                        -- are terminated.

for    TaskState      use (Initial => 16#00# ,
                           Engaged => 16#08# ,
                           Running => 16#10# ,
                           Delayed => 16#18# ,
                           EntryCallingTimed => 16#20# ,
                           EntryCallingUnconditional => 16#28# ,
                           SelectingTimed => 16#31# ,
                           SelectingUnconditional => 16#39# ,
                           SelectingTerminable => 16#41# ,
                           Accepting => 16#4A# ,
                           Synchronizing => 16#53# ,
                           Completed => 16#5C# ,
                           Terminated => 16#64#);

for    TaskState'size       use 8;

type TaskTypeDescriptor is
   record
```

```
         priority           : System.Priority;
         entry_count        : UIntg;
         block_id           : BlockId;
         first_own_address  : System.Address;
         module_number      : UIntg;
         entry_number       : UIntg;
         code_address       : System.Address;
         stack_size         : System.DWord;
         dummy              : Integer;
         stack_segment_size: UIntg;
      end record;

   for TaskTypeDescriptor use
      record
         priority           at  0 range 0..31;
         entry_count        at  2 range 0..31;
         block_id           at  4 range 0..31;
         first_own_address  at  6 range 0..31;
         module_number      at  8 range 0..31;
         entry_number       at 10 range 0..31;
         code_address       at 12 range 0..31;
         stack_size         at 14 range 0..31;
         dummy              at 16 range 0..31;
         stack_segment_size at 18 range 0..31;
      end record;

   type AccTaskTypeDescriptor is access TaskTypeDescriptor;

   type NPXSaveArea is array(1..54) of System.UnsignedWord;

pragma page;
   type TaskControlBlock is
      record
         sem    : Tasktypes.Semaphore;  -- Should be system.semaphore
                                        -- but 4.2 version of
                                        -- system is still used

      -- Delay queue handling

         dnext          : System.TaskValue ;
         dprev          : System.TaskValue ;
         ddelay         : Ticks ;

      -- Saved registers

         SS             : System.UnsignedWord ;
         SP             : Offset ;

      -- Ready queue handling

         next           : System.TaskValue ;
```

```
-- Semaphore handling

   semnext          : System.TaskValue ;

-- Priority fields

   priority         : System.Priority;
   saved_priority   : System.Priority;

-- Miscelleanous fields

   time_slice       : System.UnsignedWord;
   NPXFlag          : Bool;
   InterruptFlag    : Bool;
   ReadyCount       : System.Word;

-- Stack Specification

   stack_start      : Offset;
   stack_end        : Offset;

-- State fields

   state            : TaskState;
   is_abnormal      : Bool;
   is_activated     : Bool;
   failure          : Bool;

-- Activation handling fields

   activator        : System.TaskValue;
   act_chain        : System.TaskValue;
   next_chain       : System.TaskValue;
   no_not_act       : System.Word;
   act_block        : BlockId;

-- Accept queue fields

   partner          : System.TaskValue;
   next_partner     : System.TaskValue;

-- Entry queue fields

   next_caller      : System.TaskValue;

-- Rendezvous fields

   called_task      : System.TaskValue;
   task_entry       : TaskEntry;
   entry_index      : EntryIndex;
   entry_assoc      : System.Address;
   call_params      : System.Address;
```

```
      alt_id          : AlternativeId;
      excp_id         : System.ExceptionId;

   -- Dependency fields

      parent_task     : System.TaskValue;
      parent_block    : BlockId;
      child_task      : System.TaskValue;
      next_child      : System.TaskValue;
      first_child     : System.TaskValue;
      prev_child      : System.TaskValue;
      child_act       : System.Word;
      block_act       : System.Word;
      terminated_task: System.TaskValue;

   -- Abortion handling fields

      busy            : System.Word;

   -- Auxiliary fields

      ttd             : AccTaskTypeDescriptor;
      FirstCaller     : System.TaskValue;

   -- Run-Time System fields

      ACF              : System.UnsignedWord;  -- cf. User's Guide
                                               -- 9.4.2
      collection      : System.Address;

   -- NPX save area
   --
   -- When the application is linked with /NPX, a special
   -- save area for the NPX is allocated at the very end
   -- of every TCB.
   -- ie:
   --
   --   case NPX_Present is
   --     when TRUE  => NPXsave    : NPXSaveArea;
   --     when FALSE => null;
   --   end case;

   end record;

for TaskControlBlock use
   record
      sem              at  0 range 0..95;
      dnext            at  6 range 0..31;
      dprev            at  8 range 0..31;
      ddelay           at 10 range 0..31;
      SS               at 12 range 0..15;
      SP               at 13 range 0..31;
```

```
            next              at 15 range 0..31;
            semnext           at 17 range 0..31;
            priority          at 19 range 0..31;
            saved_priority    at 21 range 0..31;
            time_slice        at 23 range 0..15;
            NPXFlag           at 24 range 0..7;
            InterruptFlag     at 24 range 8..15;
            ReadyCount        at 25 range 0..15;
            stack_start       at 26 range 0..31;
            stack_end         at 28 range 0..31;
            state             at 30 range 0..7;
            is_abnormal       at 30 range 8..15;
            is_activated      at 31 range 0..7;
            failure           at 31 range 8..15;
            activator         at 32 range 0..31;
            act_chain         at 34 range 0..31;
            next_chain        at 36 range 0..31;
            no_not_act        at 38 range 0..15;
            act_block         at 39 range 0..31;
            partner           at 41 range 0..31;
            next_partner      at 43 range 0..31;
            next_caller       at 45 range 0..31;
            called_task       at 47 range 0..31;
            task_entry        at 49 range 0..31;
            entry_index       at 51 range 0..31;
            entry_assoc       at 53 range 0..31;
            call_params       at 55 range 0..31;
            alt_id            at 57 range 0..31;
            excp_id           at 59 range 0..63;
            parent_task       at 63 range 0..31;
            parent_block      at 65 range 0..31;
            child_task        at 67 range 0..31;
            next_child        at 69 range 0..31;
            first_child       at 71 range 0..31;
            prev_child        at 73 range 0..31;
            child_act         at 75 range 0..15;
            block_act         at 76 range 0..15;
            terminated_task   at 77 range 0..31;
            busy              at 79 range 0..15;
            ttd               at 80 range 0..31;
            FirstCaller       at 82 range 0..31;
            ACF               at 84 range 0..31;
            collection        at 86 range 0..31;
        end record;

end TaskTypes;
```

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

$ACC_SIZE An integer literal whose value is the number of bits sufficient to hold any value of an access type.	32
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	1..125 => 'A', 126 => '1'
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	1..125 => 'A', 126 => '2'
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	1..63 => 'A', 64 => '3', 65..126 => 'A'
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	1..63 => 'A', 64 => '4', 65..126 => 'A'
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	1..123 => 0, 124..126 => 298
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the	1..121 => 0, 122..126 => 690.0

maximum line length.

$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	1..63 => 'A'
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	1..62 => 'A', 63 => '1'
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	1..106 => ' '
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2147483647
$DEFAULT_MEM_SIZE An integer literal whose value is SYSTEM.MEMORY_SIZE.	16#100000000#
$DEFAULT_STOR_UNIT An integer literal whose value is SYSTEM.STORAGE_UNIT.	16
$DEFAULT_SYS_NAME The value of the constant SYSTEM.SYSTEM_NAME.	IAPX386_FM
$DELTA_DOC A real literal whose value is SYSTEM.FINE_DELTA.	2#1.0#E-31
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	35
$FIXED_NAME The name of a predefined fixed-point type other than DURATION.	NO_SUCH_TYPE
$FLOAT_NAME The name of a predefined floating-point type other than	NO_SUCH_TYPE

FLOAT, SHORT_FLOAT, or LONG_FLOAT.

$GREATER_THAN_DURATION — 100000.0
A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.

$GREATER_THAN_DURATION_BASE_LAST — 200000.0
A universal real literal that is greater than DURATION'BASE'LAST.

$HIGH_PRIORITY — 31
An integer literal whose value is the upper bound of the range for the subtype SYSTEM.PRIORITY.

$ILLEGAL_EXTERNAL_FILE_NAME1 — ILLEGAL/!*/@#$%^
An external file name which contains invalid characters.

$ILLEGAL_EXTERNAL_FILE_NAME2 — ILLEGAL&(*)_+=
An external file name which is too long.

$INTEGER_FIRST — -2147483648
A universal integer literal whose value is INTEGER'FIRST.

$INTEGER_LAST — 2147483647
A universal integer literal whose value is INTEGER'LAST.

$INTEGER_LAST_PLUS_1 — 2147483648
A universal integer literal whose value is INTEGER'LAST + 1.

$LESS_THAN_DURATION — -100000.0
A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.

$LESS_THAN_DURATION_BASE_FIRST — -200000.0
A universal real literal that is less than DURATION'BASE'FIRST.

$LOW_PRIORITY — 0
An integer literal whose value is the lower bound of the range for the subtype SYSTEM.PRIORITY.

$MANTISSA_DOC An integer literal whose value is SYSTEM.MAX_MANTISSA.	31
$MAX_DIGITS Maximum digits supported for floating-point types.	15
$MAX_IN_LEN Maximum input line length permitted by the implementation.	126
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	9223372036854775807
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	9223372036854775808
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	1..2 => '2:', 3..123 => '0', 124..126 => '11:'
$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	1..3 => '16:', 4..122 => '0', 123..126 => 'F.E:'
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	1 => '"', 2..125 => 'A', 126 => '"'
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-9223372036854775808
$MIN_TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has no entries, no declarations, and "NULL;" as the only statement in its body.	32
$NAME	NO_SUCH_TYPE

A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.

$NAME_LIST — IAPX386_FM
A list of enumeration literals in the type SYSTEM.NAME, separated by commas.

$NEG_BASED_INT — 16#FFFFFFFFFFFFFFFF#
A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.

$NEW_MEM_SIZE — 16#100000000#
An integer literal whose value is a permitted argument for pragma memory_size, other than $DEFAULT_MEM_SIZE. If there is no other value, then use $DEFAULT_MEM_SIZE.

$NEW_STOR_UNIT — 16
An integer literal whose value is a permitted argument for pragma storage_unit, other than $DEFAULT_STOR_UNIT. If there is no other permitted value, then use value of SYSTEM.STORAGE_UNIT.

$NEW_SYS_NAME — IAPX386_FM
A value of the type SYSTEM.NAME, other than $DEFAULT_SYS_NAME. If there is only one value of that type, then use that value.

$TASK_SIZE — 32
An integer literal whose value is the number of bits required to hold a task object which has a single entry with one inout parameter.

$TICK — 0.000_000_062_5
A real literal whose value is SYSTEM.TICK.

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 44 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form AI-ddddd is to an Ada Commentary.

A39005G
This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).

B97102E
This test contains an unintended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).

C97116A
This test contains race conditions, and it assumes that guards are evaluated indivisibly. A conforming implementation may use interleaved execution in such a way that the evaluation of the guards at lines 50 & 54 and the execution of task CHANGING_OF_THE_GUARD results in a call to REPORT.FAILED at one of lines 52 or 56.

BC3009B
This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).

CD2A62D
This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

CD2A63A..D, CD2A66A..D, CD2A73A..D, CD2A76A..D [16 tests]
These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD2A81G, CD2A83G, CD2A84M & N, & CD50110
These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86 & 96, 86 & 96, and 58, resp.).

CD2B15C & CD7205C
These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.

CD2D11B
This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.

CD5007B
This test wrongly expects an implicitly declared subprogram to be at the address that is specified for an unrelated subprogram (line 303).

ED7004B, ED7005C & D, ED7006C & D [5 tests]
These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.

CD7105A
This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK -- particular instances of change may be less (line 29).

CD7203B, & CD7204B
These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

CD7205D
This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

CE2107I
This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid. (line 90)

CE3111C
This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.

CE3301A
This test contains several calls to END_OF_LINE & END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, & 136).

CE3411B
This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.

E28005C
This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this that must appear at the top of the page.

APPENDIX E

COMPILER OPTIONS AS SUPPLIED BY

DDC-I, Inc

Compiler: DACS-386/UNIX, Version 4.4

ACVC Version: 1.10

OPTION	EFFECT

One invokes the Ada compiler at the UNIX shell with the following command:

$ ada {<option>} <source-file>
where <option> is:

-c <file>	Specifies the configuration file.
-d	Generates information for the DDC-I Symbolic Ada Debugger.
-e <file>	Directs error messages to specified file.
l<library>	Specifies program library used.
-L	Generates list file.
-n	Suppresses run-time checks.
-s	Copies Ada source text to program library (default).
-/s	Does not copy Ada source text to program library.
-x	Creates a cross reference listing.